Quantstamp

Startale - Paymaster

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	ERC4337 Paymaster
Timeline	2025-04-07 through 2025-04-14
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	README.md
Source Code	• StartaleLabs/scs-aa-paymasters #ce7d9b3
Auditors	• Andy Lin Senior Auditing Engineer • Ruben Koch Senior Auditing Engineer • Tim Sigl Auditing Engineer • Yamen Merhi Auditing Engineer

Documentation quality	Medium
Test quality	Medium
Total Findings	11 Fixed: 7 Acknowledged: 3 Mitigated: 1
High severity findings	1 Mitigated: 1
Medium severity findings	1 Fixed: 1
Low severity findings	4 Fixed: 3 Acknowledged: 1
Undetermined severity findings	0
Informational findings	5 Fixed: 3 Acknowledged: 2

Summary of Findings

In this audit, we reviewed two ERC-4337 Paymaster implementations compatible with EntryPoint v0.7: `SponsorshipPaymaster` and `StartaleTokenPaymaster`. The `SponsorshipPaymaster` enables native token-based sponsorships from designated sponsor entities, with verification performed through signature checks. The `StartaleTokenPaymaster` provides ERC20-based sponsorships with two operational modes: an `INDEPENDENT` mode, where token prices are fetched from oracles configurable by the owner, and an `EXTERNAL` mode, where a precomputed exchange rate is supplied alongside a signature from a privileged signer.

One High severity issue was discovered (STPM-1), related to the Paymaster possibly being drained by submitting UserOperations that fail to pay the Paymaster the needed ERC-20 token that covers what it needs to pay to the EntryPoint. Furthermore, a Medium severity issue (STPM-2) was found, where the L2 sequencer uptime is not checked within the oracle, which can cause the Paymaster to charge users based on outdated prices. Other minor issues include accounting issues, oracle issues, validation inconsistencies, and gas estimation inaccuracies.

Fix-Review Update 2025-04-29:
All issues identified during the audit have been either fully addressed or formally acknowledged. STPM-1 have been mitigated with a partial fixe; see the issue description for details.

ID	DESCRIPTION	SEVERITY	STATUS
STPM-1	`StartaleTokenPaymaster` Can Be Drained in `INDEPENDENT` Mode via Failed Token Transfers	High	Mitigated
STPM-2	Missing L2 Sequencer Uptime Check in Price Fetching Logic	Medium	Fixed
STPM-3	Rigid Oracle Decimal Mismatch Handling	Low	Fixed
STPM-4	Inconsistent Handling of Zero Exchange Rate in `EXTERNAL` Mode	Low	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
STPM-5	Unhandled Native Oracle Revert May Cause Denial of Service	Low	Fixed
STPM-6	Accounting Mismatch Between `SponsorshipPaymaster`'s Balance Tracking and `EntryPoint` Deposits	Low	Acknowledged
STPM-7	Consider Adding a Circuit Breaker to `PriceOracleHelper`	Informational	Acknowledged
STPM-8	Potential `GREP-040` Violations Leading to Paymaster Ban	Informational	Acknowledged
STPM-9	Incorrect Penalty Calculation Leads to Overestimated Gas Costs	Informational	Fixed
STPM-10	Signature Handling Deviates From `ERC4337` Specification	Informational	Fixed
STPM-11	Missing Validations	Informational	Fixed

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

> **Disclaimer**
> Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

Files Included

The following files were in scope of the audit:

```
src
├── base
│   └── BasePaymaster.sol
├── interfaces
│   ├── IOracle.sol
│   ├── IOracleHelper.sol
│   ├── ISponsorshipPaymaster.sol
│   ├── ISponsorshipPaymasterEventsAndErrors.sol
│   ├── IStartaleTokenPaymaster.sol
│   └── IStartaleTokenPaymasterEventsAndErrors.sol
├── lib
│   ├── MultiSigners.sol
│   └── TokenPaymasterParserLib.sol
├── sponsorship
│   ├── README.md
│   └── SponsorshipPaymaster.sol
├── token
│   └── startale
│       ├── PriceOracleHelper.sol
│       ├── readme.md
│       └── StartaleTokenPaymaster.sol
└── utils
    └── SoladyOwnable.sol
```

Repo: `https://github.com/StartaleLabs/scs-aa-paymasters`

Included Paths: `src/`

Operational Considerations

- The Paymaster relies on EntryPoint v0.7+ specifications (e.g., offsets for gas limits within `paymasterAndData`). These structures differ in EntryPoint v0.6, making the Paymasters incompatible with older versions.
- The `SponsorshipPaymaster._postOp()` function calculates the cost of its own execution based on the `unaccountedGas` state variable and the effective gas price (`_actualUserOpFeePerGas`). The final charge to the sponsor includes this cost. If the owner sets `unaccountedGas` too low, the Paymaster will consistently fail to charge sponsors for the full cost of its `_postOp` work, leading to a gradual loss of funds for the Paymaster operator. Conversely, if set too high, sponsors are systematically overcharged for this portion of the gas.
- The `SponsorshipPaymaster` modifies sponsor balances within `_validatePaymasterUserOp` before execution. This operation relies on the assumption that the Paymaster is **staked**, which grants it permission under ERC-7562 (STO-031) to access its own storage during validation. Deployments where the Paymaster is unstaked might face rejection from strict bundlers adhering to ERC-7562 principles for unstaked entities.
- The `SponsorshipPaymaster.executeWithdrawal()` will withdraw at most the remaining sponsored amount. Therefore, if there is any consumption between the `requestWithdrawal()` and `executeWithdrawal()`, there is a potential that less than the originally requested amount will be withdrawn.
- We assume that the `PriceOracleHelper` correctly configs the Chainlink oracle addresses.
- The `StartaleTokenPaymaster` implementation does not support non-standard ERC20 tokens, such as fee-on-transfer tokens. Balance accounting may be incorrect if non-standard tokens are added.
- `BasePaymaster._isContract()` and `MultiSigner._isSmartContract()` rely on checking whether the address's `codesize` is non-zero to determine if the address is a contract. We want to highlight that this validation can potentially be bypassed during a contract's constructor phase. Additionally, under ERC-7702, delegated EOAs may be considered contracts with the current implementation.
- Sponsors deposit ETH to the Paymaster to cover gas fees for users they wish to sponsor. While the Paymaster charges a `feeMarkup` on top of actual gas costs, sponsors are only aware of the current markup at the time of deposit. Importantly, this markup is not locked in and can be increased at any time by the Paymaster without notifying prior sponsors. As a result, sponsors may unknowingly end up covering not just gas costs but also a significantly higher premium than originally anticipated. It is essential that sponsors are made aware that the markup is dynamic and subject to change.
- In `EXTERNAL` mode, the `StartaleTokenPaymaster` charges gas fees in ERC20 tokens using pre-signed exchange rates with `validUntil` and `validAfter` parameters; however, since it doesn't limit the duration between these timestamps, users can delay transaction submission until market conditions change (e.g., a significant drop or increase in token price relative to ETH), potentially resulting in underpayment/overpayment for gas fees. We assume that the off-chain signers will validate a tighter effective period for the userOp.

Key Actors And Their Capabilities

MultiSigners Contract

- Signers: Can validate operations through their signatures

BasePaymaster Contract

- Owner: Controls deposits, withdrawals, and staking with EntryPoint

SponsorshipPaymaster Contract

- Owner: Controls parameters (min deposit, withdrawal delay, fee collector), manages signers, and can withdraw funds.
- Signers: Validate user operations for sponsorship
- Users: Can deposit, request and execute withdrawals

StartaleTokenPaymaster Contract

- Owner: Controls gas parameters, treasury address, token support, oracle configs, and can withdraw funds
- Signers: Validate user operations in `EXTERNAL` mode

PriceOracleHelper Contract

- Owner: Controls oracle configurations

Findings

STPM-1

`StartaleTokenPaymaster` Can Be Drained in `INDEPENDENT` Mode via Failed Token Transfers

High | Mitigated

Update

Marked as "Mitigated" by the client.
Addressed in: `5afe145b51d0c652fdf0dfe4597ef59a5a8ffff8`, `3bdec61b108439a533684d3586b942ca251c2f32`, `e818ef1fe14a67f928c98550482a9e1b1f78a4e6`, and `91bbeb9dc55574bd6cded9d0597c445980e8fb80`.
The client provided the following explanation:

```
We have left to implement signature check in independent mode as of now. Project may implement balance check but only bundler allowlist is added as of now which reduces the impact.
```

File(s) affected: `src/token/startale/StartaleTokenPaymaster.sol`

Description: In `INDEPENDENT` mode, the `StartaleTokenPaymaster` pays for the UserOperation's gas cost upfront by allowing the EntryPoint to deduct from its ETH deposit. It attempts to recoup this cost by transferring ERC20 tokens from the user during the `_postOp()` phase. If the user has not approved the Paymaster to spend the required tokens or has an insufficient token balance, the `_postOp()` will revert with `FailedToChargeTokens` error.

Crucially, according to the standard EntryPoint logic, a failure within `paymaster.postOp()` does not revert the gas payment. The EntryPoint proceeds to charge the Paymaster for the gas consumed up to the point of failure, plus a penalty for unused allocated gas. Additionally, in `INDEPENDENT` mode, no signature from the Paymaster's signers is required, leaving the Paymaster operator with no means to prevent user misbehavior. This allows a malicious user to repeatedly trigger this failure state, draining the Paymaster's deposited ETH balance without ever transferring the required ERC20 tokens.

Exploit Scenario:

1. Setup: An attacker targets the `StartaleTokenPaymaster` (in `INDEPENDENT` mode). They craft a UserOperation ensuring the subsequent ERC20 `transferFrom()` by the paymaster in `_postOp` will fail (either by not providing token approval or having insufficient balance).
2. Gas Inflation: The attacker intentionally includes excessively high `callGasLimit` and `paymasterPostOpGasLimit` values within the UserOperation to maximize potential gas costs and penalties.
3. Execution & Failure: The EntryPoint processes the UserOp. While the main execution might proceed, the Paymaster's `_postOp()` fails when attempting the ERC20 `transferFrom()` function.
4. Cost Borne by Paymaster: Although the token transfer failed, the EntryPoint still charges the Paymaster for the gas consumed up to the failure point. Crucially, it also adds a 10% penalty calculated on the large amount of unused gas (due to the inflated limits), further increasing the cost.
5. Drain: The attacker pays nothing, while the Paymaster loses ETH covering both the consumed gas and the penalty. By repeatedly submitting such UserOperations, the attacker can systematically drain the Paymaster's ETH deposit in the EntryPoint.

Recommendation:

1. **Pre-charge and refund**: Consider charging the expected cost during `_validatePaymasterUserOp()` (e.g., using `_requiredPreFund`) and refund any excess in `_postOp()`.
2. **Off-chain signature checks**: Add a signing mechanism to filter known malicious users, validate gas limit values, and act as a circuit breaker to pause service during suspicious activity.
3. **Whitelisted bundlers**: Only allow the `StartaleTokenPaymaster` contract to be used with whitelisted bundlers by checking against `tx.origin` to avoid attacks from unknown users.

STPM-2

Missing L2 Sequencer Uptime Check in Price Fetching Logic

Medium | Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `cd6789f3b7bcc81d2d6b6e099dd1f54a143d8cdc`, `c4e5aa6389ede08cea57174103cefb398708a19f`.

File(s) affected: `src/token/startale/PriceOracleHelper.sol`

Description: The `PriceOracleHelper.fetchPrice()` function validates Chainlink oracle data by checking the `updatedAt` timestamp and `answeredInRound`, but it does not verify the operational status of the Layer 2 (L2) sequencer. On L2s, Chainlink may return recent timestamps even if the sequencer is offline, since the oracle nodes source price data from L1. This may lead to stale prices being accepted as fresh, causing incorrect exchange rates to be used.

This is critical in the `INDEPENDENT` mode of `StartaleTokenPaymaster`, where `fetchPrice()` is used via `getExchangeRate()` during `_postOp()` to calculate token charges. Without the sequencer check, the Paymaster may charge users based on outdated prices, risking loss of funds or unfair overcharging.

Recommendation: Implement a check for L2 sequencer uptime status using Chainlink's L2 Sequencer Uptime Feed. Abort the price fetch if the sequencer is reported as down or if the grace period after restart has not elapsed.

STPM-3 Rigid Oracle Decimal Mismatch Handling

• Low Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `791d560d5222469446091a23739d8909138a0524`.

File(s) affected: `src/token/startale/PriceOracleHelper.sol`

Description: The Chainlink documentation states that "Each feed uses a different number of decimal places for answers". The strict guard in `getExchangeRate()` with the condition `if (IOracle(config.tokenOracle).decimals() != IOracle(nativeAssetToUsdOracle).decimals())` may lead to incompatibility with certain feeds. A better approach is to convert both values to the higher decimal precision and perform calculations accordingly.

Recommendation: Allow mismatched decimals and normalize prices by scaling to the higher decimals, as follows:

```
uint8 tokenOracleDecimals = IOracle(config.tokenOracle).decimals();
uint8 nativeOracleDecimals = IOracle(nativeAssetToUsdOracle).decimals();
uint256 tokenPrice = fetchPrice(config.tokenOracle, config.maxOracleRoundAge);
uint256 nativePrice = fetchPrice(nativeAssetToUsdOracle, nativeOracleConfig.maxOracleRoundAge);

if (tokenOracleDecimals > nativeOracleDecimals) {
    nativePrice *= 10 ** (tokenOracleDecimals - nativeOracleDecimals);
} else if (tokenOracleDecimals < nativeOracleDecimals) {
    tokenPrice *= 10 ** (nativeOracleDecimals - tokenOracleDecimals);
}

exchangeRate = (nativePrice * 10 ** IERC20Metadata(_token).decimals()) / tokenPrice;
```

STPM-4 Inconsistent Handling of Zero Exchange Rate in `EXTERNAL` Mode

• Low Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `a34ca793a7805998e4d8c881722bf3fffe04d1a4`.

File(s) affected: `src/token/startale/StartaleTokenPaymaster.sol`

Description: In the `EXTERNAL` mode branch of `_validatePaymasterUserOp()`, the `exchangeRate` is obtained from the signed message. However, the function does not validate that `exchangeRate` is non-zero. In contrast to the `INDEPENDENT` mode—where a zero `exchangeRate` triggers a fallback to retrieve the price from the oracle in `_postOp()`—a zero `exchangeRate` in `EXTERNAL` mode will still be used for signature verification and subsequently replaced with the oracle price in the post-operation phase.

This discrepancy can lead to a mismatch between the value originally signed by the off-chain signer and the final exchange rate used for fee calculation.

Recommendation: Enforce a validation check in the `EXTERNAL` mode branch to ensure that the provided `exchangeRate` is non-zero.

STPM-5
Unhandled Native Oracle Revert May Cause Denial of Service

• Low ⓘ Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `24638e54954fe9c5d4da0fd1b3b8e4d74888a7dc`, `2d6193a0c3ea143450f84cb4d27d45c43e71f507`.

File(s) affected: `src/token/startale/PriceOracleHelper.sol`

Description: The `PriceOracleHelper` contract relies on the `nativeAssetToUsdOracle` to retrieve the native asset's price (e.g., ETH/USD). This oracle is configured once at deployment and cannot be updated post-deployment. In contrast to token oracles, which are upgradable via `_updateTokenOracleConfig()`, the native oracle has no upgrade mechanism.

All token-to-native asset conversions use the `getExchangeRate()` function, which computes the token price relative to the native asset. Importantly, every call to `getExchangeRate()` for any token depends on a successful query to the `nativeAssetToUsdOracle`. This means a failure in the native oracle affects all token pricing logic system-wide.

If the native oracle becomes unavailable—for example, due to data feed deprecation, misconfiguration, or blacklisting by an oracle provider like Chainlink—then calls to `fetchPrice()` will revert. This results in a complete denial-of-service (DoS) for the `StartaleTokenPaymaster` or any other system using this helper, as it will be unable to compute token exchange rates, halting all core operations. Since the contract has no mechanism to update the native oracle, recovery would require full contract redeployment, leading to potential downtime, loss of state, or loss of funds unless explicitly migrated.

Exploit Scenario:
1. Chainlink disables or removes access to the ETH/USD feed used as the native oracle.
2. `fetchPrice(nativeAssetToUsdOracle)` reverts inside `getExchangeRate()`.
3. Any token pricing call fails, even if the token's individual oracle is functioning.
4. Paymaster logic fails to execute, rejecting user operations.
5. The contract is effectively bricked, with no recourse other than redeployment.

Recommendation:
- Introduce a function with access control to allow updating the native oracle feed.
- Consider wrapping oracle calls in `try/catch` to allow for controlled fallback behavior e.g. with a second oracle used as fallback.
- Monitor the operational health of the native oracle and prepare contingency procedures for migrating or replacing it.

STPM-6
Accounting Mismatch Between `SponsorshipPaymaster`'s Balance Tracking and `EntryPoint` Deposits

• Low ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
given the less likelihood and multiple dapps contributing to the balance, capping does not seem
necessary.
```

File(s) affected: `src/sponsorship/SponsorshipPaymaster.sol`

Description: The `SponsorshipPaymaster` contract maintains a separate accounting system (`sponsorBalances`) to track sponsor funds, but this system can gradually diverge from the actual ETH deposited in the EntryPoint contract. Although the calculations in `SponsorshipPaymaster` aim to mimic the value held in the EntryPoint, there is still a non-zero chance of reconciliation mismatches, where sponsorBalances do not deduct enough to cover the gas fees paid to the bundler:

1. The `unaccountedGas` does not represent the exact gas used in `postOp()`, plus the gas consumed by the EntryPoint itself. When `unaccountedGas` is less than the actual usage, it results in insufficient deductions from the `sponsorBalance`.
2. The `preOpGasApproximation` can, in rare case, slightly underestimate actual gas usage. For instance, if `validateUserOp()` and `validatePaymasterUserOp()` use exactly the provided gas limit, the extra gas used by the EntryPoint is not accounted for in `preOpGasApproximation`. This can result in a slightly higher `executionGasUsed`, reducing the `expectedPenaltyGas` in `_postOp()`, and causing the penalty to be undercharged.

If the `sponsorBalance` is deducted by less than the actual amount paid from the Paymaster deposit, it can cause `executeWithdrawal()` to fail unexpectedly when calling `entryPoint.withdrawTo(payable(req.to), req.amount)`.

Recommendation: While full reconciliation with the EntryPoint deposit may be difficult—and in most natural cases, the `SponsorshipPaymaster` will slightly overcharge rather than undercharge, aside from the edge cases described—we recommend updating the

`executeWithdrawal()` function to ensure it can still execute withdrawals even when the EntryPoint balance is insufficient. Consider capping `req.amount` to `entryPoint.balanceOf(address(this))`.

STPM-7
Consider Adding a Circuit Breaker to `PriceOracleHelper`

• Informational Acknowledged

> **Update**
> Marked as "Acknowledged" by the client.
> The client provided the following explanation:
>
> `we could de-list the token or stop producing signatures.`

File(s) affected: `src/token/startale/PriceOracleHelper.sol`

Description: The contract lacks a fallback mechanism in case the primary oracle fails or returns extremely unusual values. If an oracle is compromised or malfunctions, the system may continue using potentially manipulated prices until manually updated.

Recommendation: Implement a circuit breaker mechanism that can pause affected operations when abnormal price movements or oracle failures are detected. Possible approaches include:

1. **Admin-based pausing mechanism**: Add a pausing feature to the contract that allows administrators to pause oracle usage when significant price deviations are detected, mitigating further risk. Note that in this case, the pausing mechanism should ideally be checked on the validation stage instead of `postOp()`.
2. **On-chain price deviation detector**: Track the last-seen price and the current price. If the new price deviates significantly from the previous one within a short time frame, this could indicate a potential oracle failure and should trigger the circuit breaker.

STPM-8
Potential `GREP-040` Violations Leading to Paymaster Ban

• Informational Acknowledged

> **Update**
> Marked as "Acknowledged" by the client.
> The client provided the following explanation:
>
> `there is no direct fix. But we are aware of this behaviour.`

File(s) affected: `src/sponsorship/SponsorshipPaymaster.sol`, `src/token/startale/StartaleTokenPaymaster.sol`

Description: The `_validatePaymasterUserOp()` function includes state-dependent reverts that might change in outcome between second validation and final simulation/execution on the chain, such as `unaccountedGas` being updated, updates to the supported tokens and updates to the signers and withdrawal requests by sponsors. If validation produces different results for the same UserOperation after the second validation simulation, the EntryPoint may ban the Paymaster. This aligns with GREP-40 from ERC-7562, which emphasizes deterministic validation to prevent bans. Here is a non-exhaustive list of where this might occur:

1. The `setUnaccountedGas()` function can update the required lower bound for `postOpGasLimits`. Depending on when the function is called, there can be a case where UserOperation with a tight `postOpGasLimit` is invalidated just before on-chain execution.
2. Signers can be removed via `removeSigner()`, which could lead to signature validation failing and there can be a case where UserOperation is invalidated just before on-chain execution.
3. Support for tokens can be revoked via `StartaleTokenPaymaster.removeSupportedToken()`, which would could create validation execution inconsistencies due to the check for support in the Paymaster validation.
4. A sponsor can execute a withdrawal request while being the supposed sponsor for pending UserOperations, potentially causing underflows in the balance deduction after second validation.

Recommendation: There is no direct fix, but it is important to be aware of this behavior. Consider documenting this risk and monitoring for unexpected Paymaster bans. Calls to variable increases via `setUnaccountedGas()` and calls to `removeSigner()` and `removeSupportedToken()` should be done with a temporary maintenance pause of the signing process, or carefully set these values leading up to such events. Potential sponsors with (imminently) executable withdrawals should only be used as sponsors if their post-withdrawal-execution balance is sufficient.

STPM-9
Incorrect Penalty Calculation Leads to Overestimated Gas Costs

• Informational Fixed

> **Update**

Marked as "Fixed" by the client.
Addressed in: `cafb1ba9cc229a11bc31b6d68f6d3cf438360948`.

File(s) affected: `src/sponsorship/SponsorshipPaymaster.sol`, `src/token/startale/StartaleTokenPaymaster.sol`

Description: The EntryPoint v0.7 contract includes a penalty calculation for unused execution gas. These Paymasters forward the payment of that one to the user; however, the penalty is incorrectly calculated. The penalty is not added to on top of the upper bound estimates of the gas limits as additional costs. Instead, the penalty is only a share from the difference between allocated gas and used gas that is not returned to the entity prefunding the operation. The rest of the excessive gas (so the penalty-reduced, unused gas that was overcharged from the prefunding), is returned to the prefunding entity. Therefore, for the preCharge calculations in the validation phase, no penalty calculations need to happen. For reference, see bullet point about the penalty in this section of ERC-4337.

The only impact is that the sponsors are slightly overcharged initially, the funds are properly returned to them in the `postOp()` flow.

Recommendation: Remove the penalty calculation from the Paymaster validation logic.

STPM-10 Signature Handling Deviates From `ERC4337` Specification

• Informational Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `391204930f2401df1571c95f539ed17f92e3023b`.

File(s) affected: `src/token/startale/StartaleTokenPaymaster.sol`, `src/sponsorship/SponsorshipPaymaster.sol`

Description: In the `_validatePaymasterUserOp()` function, the signature is currently verified using `ECDSA.tryRecover()`. it This function does not revert for malformed signatures but returns `address(0)` instead. According to ERC-4337, validation logic is expected to revert when presented with invalid (malformed) signatures, and only return `SIG_VALIDATION_FAILED` for cases where the signature is well-formed but incorrect.

Recommendation: Consider replacing `ECDSA.tryRecover()` with `ECDSA.recover()` from the Solady library. This ensures malformed signatures are rejected through a revert, as required by ERC-4337.

STPM-11 Missing Validations

• Informational Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `16c0ed9c360933f640d098968afeab4a76ad7c95`, `55f2bd8c2e3d15af1c4e1547a4007cf9a38314f6`.

File(s) affected: `src/token/startale/PriceOracleHelper.sol`, `src/sponsorship/SponsorshipPaymaster.sol`, `src/token/startale/StartaleTokenPaymaster.sol`

Description: Certain aspects of the code can benefit from extra validation:

1. `TokenPaymasterParserLib.parsePaymasterAndData()`: consider adding validation to explicitly revert when `_paymasterAndData.length < PAYMASTER_MODE_OFFSET + 1`.
2. `TokenPaymasterParserLib.parseIndependentModeSpecificData()`: consider adding validation to explicitly revert when `modeSpecificData.length` < 20.
3. `TokenPaymasterParserLib.parseExternalModeSpecificData()`: consider adding validation to explicitly revert when `modeSpecificData.length` < 70.
4. `SponsorshipPaymaster.setMinDeposit()`: consider adding validation to ensure that `_newMinDeposit` cannot be zero, or set a hardcoded minimum value here.
5. `PriceOracleHelper.constructor()`: consider adding an upper bound check here to ensure that `_tokenOracleConfigs[i].maxOracleRoundAge <= MAX_ALLOWED_ROUND_AGE`.
6. `SponsorshipPaymaster .setWithdrawalDelay()`: consider adding an upper bound check to ensure that the `sponsorWithdrawalDelay` cannot be a high value disallowing sponsors to withdraw their deposits.
7. `SponsorshipPaymaster.withdrawEth()` and `StartaleTokenPaymaster.withdrawEth()`: consider validating that the recipient is not the zero address to avoid accidental burning of ETH.
8. `PriceOracleHelper._updateNativeOracleConfig()`: Consider checking that `NativeOracleConfig.maxOracleRoundAge != 0`.
9. `MultiSigner._addSigner()` and `MultiSigner._removeSigner()`: Consider checking if a signer is already added or is already removed before emitting the corresponding events.

Recommendation: Consider adding the extra validation.

Auditor Suggestions

S1 Weak Enforcement of `minDeposit`

Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `cba7f1c75a7d770b05b3b36ba30e1b264b2ea7a1`.

File(s) affected: `src/sponsorship/SponsorshipPaymaster.sol`

Description: The `minDeposit` variable is supposed to enforce that a Paymaster deposits not less than some amount initially. However, a sponsor can technically deposit above `minDeposit` and proceed to withdraw with some value to fall below `minDeposit` right after to avoid `minDeposit` restrictions.

It should be noted that a sponsor's balance can of course fall below `minDeposit` through sponsored `UserOperations` reducing their balance sufficiently, but that is of course a very acceptable case of falling below that `minDeposit` threshold.

Recommendation: Consider reverting unless the entire balance is withdrawn when a withdrawal would cause the sponsor's balance to fall below `minDeposit`.

S2 No Standard Way for Withdrawal Request Cancellation

Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `b64864c202c9f7a6630f47d40846047498f765e2`.

File(s) affected: `src/sponsorship/SponsorshipPaymaster.sol`

Description: The `requestWithdrawal()` function allows sponsors to request withdrawals from their balance but does not provide a standard method to cancel these withdrawal requests. Currently, a user can only indirectly cancel withdrawal requests by calling `requestWithdrawal()` with a smaller withdrawal. The sponsor can be frontrunned by an actor who execute the withdrawal, while the sponsor makes another call to `requestWithdrawal()` with a smaller amount.

Recommendation: Implement a dedicated `cancelWithdrawal()` function or a standardized cancellation mechanism.

S3 Application Monitoring Can Be Improved by Emitting More Events

Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `6c2556d84f5ecf79dc43857cbfe7f88114272cc3`.

File(s) affected: `src/sponsorship/SponsorshipPaymaster.sol`, `src/token/startale/StartaleTokenPaymaster.sol`, `src/token/startale/PriceOracleHelper.sol`, `src/lib/MultiSigners.sol`

Description: The following updates of contract's state do not result in emitting an event, making it harder to track contract state changes (intended or not):

- Initial values of `feeCollector`, `minDeposit`, `sponsorWithdrawalDelay` and `unaccountedGas` in `SponsorshipPaymaster.constructor()`.
- `sponsorWithdrawalDelay` in `setWithdrawalDelay()`.
- Initial values of `tokenOracleConfigurations[token]` in `PriceOracleHelper.constructor()`.
- Initial values of `tokenFeesTreasury`, and `unaccountedGas` in `StartaleTokenPaymaster.constructor()`.
- `unaccountedGas` in `setUnaccountedGas()` in `StartaleTokenPaymaster` contract.
- `tokenFeesTreasury` in `setTokenFeesTreasury()` in `StartaleTokenPaymaster` contract.
- Initial values of `signers` set in the `MultiSigners.constructor()`.

Recommendation: Consider emitting events for state changes.

S4 Perform Explicit Rounding in Favour of Paymaster

Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `fc736165c69e7851eab7650f8e9588ee06ec3c86`.

File(s) affected: `src/token/startale/StartaleTokenPaymaster.sol`, `src/sponsorship/SponsorshipPaymaster.sol`

Description: Implicit rounding as part of divisions should be performed in favour of the Paymaster entity. The following cases have been identified:

- In both `_postOp()` calls, `adjustedGasCost` should be explicitly rounded up.
- In `StartaleTokenPaymaster._postOp()`, `tokenAmount` should be explicitly rounded up.
- In `SponsorshipPaymaster._validatePaymasterUserOp()`, `effectiveCost` should be explicitly rounded up.

Recommendation: Consider implementing these explicit roundings.

S5 Remove Unused, Incorrect `parsePaymasterAndDataForExternalMode()` Function

Acknowledged

Update
Marked as "Unresolved" by the client.
The client provided the following explanation:

```
have kept it as discussed it could be useful for debugging purposes off-chain.
```

File(s) affected: `src/token/startale/StartaleTokenPaymaster.sol`

Description: The public `parsePaymasterAndDataForExternalMode()` function seems to be an artefact of prior development iterations. The function is not only unused, but its decoded format does not match the current context-encoding of the `EXTERNAL` mode.

Recommendation: Remove the unused function and its associated test.

S6 One-Step Ownership Transfers and Ownership Renouncement Should Be Disabled

Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `a61de8bce091633825165e139dfcd38f10fd2d02`.

File(s) affected: `src/utils/SoladyOwnable.sol`

Description: The `Ownable` contract from Solady provides both one-step (`transferOwnership()`) and two-step (`requestOwnershipHandover()` → `completeOwnershipHandover()`) ownership transfer mechanisms. While this offers flexibility, enabling direct transfers with `transferOwnership()` can lead to accidental or unauthorized ownership changes, particularly in high-value contracts.

Additionally, the `renounceOwnership()` function allows the current owner to renounce ownership, leaving contract functions inaccessible. In scenarios where continued administrative control is required, renouncement should be explicitly disallowed.

Recommendation:

1. Override `transferOwnership()` in any inheriting contract to always revert. This enforces exclusive use of the safer two-step handover process (`requestOwnershipHandover()` followed by `completeOwnershipHandover()`) or use the `Ownable2Step` pattern from OpenZeppelin.
2. Override `renounceOwnership()` to revert if ownership renouncement is not desired.

S7 Avoid Transfer on Zero Amount

Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `89ccb95cefb6d531a59948515937249a952a453e`.

File(s) affected: `src/token/startale/StartaleTokenPaymaster.sol`

Description: The `_postOp()` function can potentially attempt to transfer zero tokens in the `SafeTransferLib.trySafeTransferFrom()` call. For instance, when the `appliedFeeMarkup` is set to zero, the resulting `tokenAmount` to transfer will also be zero.

While this is acceptable for most token implementations, there is a risk that some tokens may implement a sanity check against zero transfers, causing a revert. If the ERC20 transfer reverts, it will trigger the `FailedToChargeTokens` error in the `_postOp()` function, leading to the complete reversion of the userOp execution, even though the Paymaster is still required to pay for the gas cost.

Recommendation: Consider avoiding the call to `trySafeTransferFrom()` when the amount of tokens is zero.

S8 Confusing Error Messages in Oracle Validation

Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `2b20e072b3391db7c0908fcafefa7cd3c819dd69`.

File(s) affected: `src/token/startale/PriceOracleHelper.sol`

Description: The fetchPrice function uses error messages that do not accurately reflect the conditions being checked.

- The staleness check (`updatedAt < block.timestamp - _maxOracleRoundAge`) incorrectly reverts with `IncompleteRound`.
- The round completeness check (`answeredInRound < roundId`) incorrectly reverts with `StalePrice`.

These misleading messages make debugging difficult and may lead to misinterpretation of why an oracle price fetch failed, potentially hindering operational responses.

Recommendation: Update error messages to match the condition being checked:

- The staleness check should revert with `StalePrice`.
- The round completeness check should revert with `IncompleteRound`.

S9 General Suggestions & Best Practices

Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `7d05bd39577f5f205a773159fd417a7e4faee7dc`, `b2f0be9570b28dd3b245d7af27808b8061c6ae28`.

File(s) affected: `src/token/startale/PriceOracleHelper.sol`, `src/sponsorship/SponsorshipPaymaster.sol`, `src/lib/MultiSigners.sol`

Description:

1. `MultiSigners` contract: Remove the unused import `IEntryPoint`.
2. `MultiSigners` contract: Consider adding an `indexed` parameter to the `SignerAdded` and `SignerRemoved` events to improve event filtering.
3. `MultiSigners.constructor()`: Using `unchecked {++i}` in a `for` loop no longer provides gas savings as of Solidity v0.8.22. Consider switching to the standard `for (uint256 i; i < length; ++i)` syntax.
4. `SponsorshipPaymaster`: Several admin functions (`setFeeCollector()`, `addSigner()`, `removeSigner()`, `setUnaccountedGas()`, `withdrawEth()`) are marked `payable` but do not use `msg.value`. Consider removing the `payable` keyword.
5. `SponsorshipPaymaster._validatePaymasterUserOp()`: In the `maxPenalty` calculation, consider replacing `uint128(uint256(_userOp.accountGasLimits))` with `_userOp.unpackCallGasLimit()`, and `_userOp.paymasterAndData[PAYMASTER_POSTOP_GAS_OFFSET:PAYMASTER_DATA_OFFSET]` with `_userOp.unpackPostOpGasLimit()` for improved readability.
6. `SponsorshipPaymaster._validatePaymasterUserOp()`: Consider replacing the magic number `10` with a named constant (e.g., `PENALTY_PERCENT`) for clarity and consistency, similar to the EntryPoint contract.
7. `PriceOracleHelper.fetchPrice()`: according to the Chainlink doc, the `answeredInRound` is deprecated in aggregator v3. Consider removing the validation that reverts on `answeredInRound < roundId`.
8. The `MultSigners._addSigner()` and `MultSigners._removeSigner()` functions are marked `virtual` despite not being overwritten in any inheriting contract.

Recommendation: Consider applying the suggestions mentioned above.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.

- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Files

Repo: `https://github.com/StartaleLabs/scs-aa-paymasters`

- `2e3...589 ./src/base/BasePaymaster.sol`
- `5c4...091 ./src/interfaces/IOracle.sol`
- `001...10a ./src/interfaces/IOracleHelper.sol`
- `a2f...776 ./src/interfaces/ISponsorshipPaymaster.sol`
- `ac4...a77 ./src/interfaces/ISponsorshipPaymasterEventsAndErrors.sol`
- `d4e...7d7 ./src/interfaces/IStartaleTokenPaymaster.sol`
- `f1e...7e1 ./src/interfaces/IStartaleTokenPaymasterEventsAndErrors.sol`
- `6b4...877 ./src/lib/MultiSigners.sol`
- `95e...caa ./src/lib/TokenPaymasterParserLib.sol`
- `c99...e2a ./src/sponsorship/README.md`
- `58a...c1f ./src/sponsorship/SponsorshipPaymaster.sol`
- `3af...037 ./src/token/startale/PriceOracleHelper.sol`
- `c66...2e2 ./src/token/startale/StartaleTokenPaymaster.sol`
- `e91...9a1 ./src/token/startale/readme.md`
- `7e7...57c ./src/utils/SoladyOwnable.sol`

Files

- `001...10a ./src/interfaces/IOracleHelper.sol`
- `ac4...a77 ./src/interfaces/ISponsorshipPaymasterEventsAndErrors.sol`
- `f1e...7e1 ./src/interfaces/IStartaleTokenPaymasterEventsAndErrors.sol`
- `5c4...091 ./src/interfaces/IOracle.sol`
- `a2f...776 ./src/interfaces/ISponsorshipPaymaster.sol`
- `d4e...7d7 ./src/interfaces/IStartaleTokenPaymaster.sol`
- `2e3...589 ./src/base/BasePaymaster.sol`
- `6b4...877 ./src/lib/MultiSigners.sol`
- `95e...caa ./src/lib/TokenPaymasterParserLib.sol`
- `58a...c1f ./src/sponsorship/SponsorshipPaymaster.sol`
- `7e7...57c ./src/utils/SoladyOwnable.sol`
- `3af...037 ./src/token/startale/PriceOracleHelper.sol`
- `c66...2e2 ./src/token/startale/StartaleTokenPaymaster.sol`

Test Suite Results

Test data output was obtained with `forge test`. All the tests ran with 91 tests executed successfully.

Fix-Review Update: The tests increased to 110 tests executed successfully.

```
Ran 13 tests for test/foundry/SponsorshipPaymaster.Old.t.sol:SponsorshipPaymasterTest
[PASS] testSponsorshipSuccess() (gas: 251150)
[PASS] test_DepositFor() (gas: 80612)
[PASS] test_RevertIf_DepositIsZero() (gas: 14744)
[PASS] test_RevertIf_SetUnaccountedGasTooHigh() (gas: 13644)
[PASS] test_RevertIf_TriesWithdrawToWithoutRequest() (gas: 13027)
[PASS] test_SetUnaccountedGas() (gas: 21484)
[PASS] test_addNewSigner() (gas: 42320)
[PASS] test_executeWithdrawalRequest_Fails_with_NoRequestSubmitted() (gas: 19129)
[PASS] test_executeWithdrawalRequest_Happy_Scenario() (gas: 163172)
[PASS] test_executeWithdrawalRequest_Reverts_If_Withdraws_TooSoon() (gas: 167452)
[PASS] test_removeSigner() (gas: 33349)
[PASS] test_submitWithdrawalRequest_Fails_with_ZeroAddress() (gas: 12037)
[PASS] test_submitWithdrawalRequest_Fails_with_ZeroAmount() (gas: 14156)
Suite result: ok. 13 passed; 0 failed; 0 skipped; finished in 2.64ms (2.85ms CPU time)

Ran 44 tests for test/foundry/unit/concrete/TestSponsorshipPaymaster.t.sol:TestSponsorshipPaymaster
[PASS] test_AddVerifyingSigner() (gas: 57057)
[PASS] test_CheckInitialPaymasterState() (gas: 38772)
[PASS] test_Deploy() (gas: 1880603)
[PASS] test_DepositFor() (gas: 84488)
[PASS] test_Failure_TwoStepOwnershipTransferExpired() (gas: 48701)
[PASS] test_Failure_TwoStepOwnershipTransferWithdrawn() (gas: 35940)
[PASS] test_OwnershipTransfer() (gas: 25313)
[PASS] test_ParsePaymasterAndData() (gas: 66292)
[PASS] test_Receive() (gas: 18531)
[PASS] test_RemoveVerifyingSigner() (gas: 28704)
[PASS] test_RevertIf_AddVerifyingSignerToZeroAddress() (gas: 27706)
[PASS] test_RevertIf_DeployWithFeeCollectorAsContract() (gas: 303118)
[PASS] test_RevertIf_DeployWithFeeCollectorSetToZero() (gas: 126820)
[PASS] test_RevertIf_DeployWithSignerAsContract() (gas: 278746)
[PASS] test_RevertIf_DeployWithSignerSetToZero() (gas: 101431)
[PASS] test_RevertIf_DeployWithUnaccountedGasCostTooHigh() (gas: 132443)
[PASS] test_RevertIf_DepositCalled() (gas: 9792)
[PASS] test_RevertIf_DepositForZeroAddress() (gas: 16804)
[PASS] test_RevertIf_DepositForZeroValue() (gas: 12777)
[PASS] test_RevertIf_OwnershipTransferToZeroAddress() (gas: 15855)
[PASS] test_RevertIf_SetFeeCollectorToZeroAddress() (gas: 14575)
[PASS] test_RevertIf_SetUnaccountedGasToHigh() (gas: 14801)
[PASS] test_RevertIf_TriesWithdrawToWithoutRequest() (gas: 14115)
[PASS] test_RevertIf_UnauthorizedOwnershipTransfer() (gas: 14639)
[PASS] test_RevertIf_UnauthorizedSetFeeCollector() (gas: 12872)
[PASS] test_RevertIf_ValidatePaymasterUserOpWithIncorrectSignatureLength() (gas: 205054)
[PASS] test_RevertIf_ValidatePaymasterUserOpWithInsufficientDeposit() (gas: 95188)
[PASS] test_RevertIf_ValidatePaymasterUserOpWithInvalidPriceMarkUp() (gas: 172948)
[PASS] test_RevertIf_WithdrawErc20ToZeroAddress() (gas: 538535)
[PASS] test_RevertIf_WithdrawEthExceedsBalance() (gas: 26578)
[PASS] test_SetFeeCollector() (gas: 27061)
[PASS] test_SetUnaccountedGas() (gas: 25838)
[PASS] test_Success_TwoStepOwnershipTransfer() (gas: 39512)
[PASS] test_ValidatePaymasterAndPostOpWithPriceMarkup() (gas: 251332)
[PASS] test_ValidatePaymasterAndPostOpWithoutPriceMarkup() (gas: 252886)
[PASS] test_WithdrawErc20() (gas: 555155)
[PASS] test_WithdrawEth() (gas: 27751)
[PASS] test_depositFor_RevertsIf_DepositIsLessThanMinDeposit() (gas: 23108)
[PASS] test_executeWithdrawalRequest_Fails_with_NoRequestSubmitted() (gas: 20129)
[PASS] test_executeWithdrawalRequest_Withdraws_WhateverIsLeft() (gas: 251420)
[PASS] test_submitWithdrawalRequest_Fails_If_not_enough_balance() (gas: 86428)
[PASS] test_submitWithdrawalRequest_Fails_with_ZeroAddress() (gas: 12601)
[PASS] test_submitWithdrawalRequest_Fails_with_ZeroAmount() (gas: 15772)
[PASS] test_submitWithdrawalRequest_Happy_Scenario() (gas: 139604)
Suite result: ok. 44 passed; 0 failed; 0 skipped; finished in 5.65ms (7.86ms CPU time)

Ran 25 tests for test/foundry/unit/concrete/TestTokenPaymaster.t.sol:TestTokenPaymaster
[PASS] testCounter() (gas: 3151)
[PASS] testToken() (gas: 2821)
```

```
[PASS] test_AddVerifyingSigner() (gas: 57475)
[PASS] test_Allow_Treasury_ToBeSelf() (gas: 2214097)
[PASS] test_Deploy_STPM() (gas: 2220023)
[PASS] test_Deposit() (gas: 58212)
[PASS] test_ParsePaymasterAndDataForExternalMode() (gas: 62504)
[PASS] test_RemoveVerifyingSigner() (gas: 28931)
[PASS] test_RevertIf_DeployWithSignerSetToZero() (gas: 110864)
[PASS] test_RevertIf_FeeTreasuryIsZero() (gas: 203194)
[PASS] test_RevertIf_InvalidSignature_ExternalMode() (gas: 231502)
[PASS] test_RevertIf_InvalidTokenAddress_Independent_Mode() (gas: 667861)
[PASS] test_RevertIf_Mismatching_Oracle_Decimals_Independent_Mode() (gas: 286712)
[PASS] test_RevertIf_UnaccountedGasTooHigh() (gas: 14653)
[PASS] test_RevertIf_UserDoesNotHaveEnoughBalance_Any_Mode() (gas: 268247)
[PASS] test_Revert_PostOp_If_PriceExpired() (gas: 289002)
[PASS] test_Revert_PostOp_If_StalePrice() (gas: 288949)
[PASS] test_SetPriceMarkupTooHigh() (gas: 19205)
[PASS] test_Success_TokenPaymaster_ExternalMode_WithPremium() (gas: 294817)
[PASS] test_Success_TokenPaymaster_ExternalMode_WithoutPremium() (gas: 298056)
[PASS] test_Success_TokenPaymaster_IndependentMode_WithPremium() (gas: 319025)
[PASS] test_Success_TokenPaymaster_IndependentMode_WithoutPremium() (gas: 312313)
[PASS] test_Success_test_UpdateNativeOracleConfig() (gas: 20095)
[PASS] test_WithdrawERC20() (gas: 68508)
[PASS] test_WithdrawTo() (gas: 54963)
Suite result: ok. 25 passed; 0 failed; 0 skipped; finished in 6.20ms (9.71ms CPU time)

Ran 4 tests for test/foundry/unit/fuzz/TestFuzz_SponsorshipPaymaster.t.sol:TestFuzz_SponsorshipPaymaster
[PASS] testFuzz_DepositFor(uint256) (runs: 1000, μ: 85462, ~: 85462)
[PASS] testFuzz_Receive(uint256) (runs: 1000, μ: 19146, ~: 19146)
[PASS] testFuzz_WithdrawErc20(address,uint256) (runs: 1000, μ: 554657, ~: 555016)
[PASS] testFuzz_WithdrawEth(uint256) (runs: 1000, μ: 27256, ~: 27256)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 185.95ms (293.60ms CPU time)

Ran 5 tests for test/foundry/unit/concrete/TestTokenPaymaster.Soneium.t.sol:TestTokenPaymasterSoneium
[PASS] testCounter() (gas: 2755)
[PASS] test_Deploy_STPM_Soneium() (gas: 2220270)
[PASS] test_Deposit_Soneium() (gas: 57723)
[PASS] test_Success_TokenPaymaster_IndependentMode_WithoutPremium_Soneium() (gas: 539900)
[PASS] test_WithdrawTo_Soneium() (gas: 54804)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 7.25s (3.01s CPU time)

Ran 5 test suites in 7.26s (7.45s CPU time): 91 tests passed, 0 failed, 0 skipped (91 total tests)

**Fix-Review Update:**

Ran 13 tests for test/foundry/SponsorshipPaymaster.Old.t.sol:SponsorshipPaymasterTest
[PASS] testSponsorshipSuccess() (gas: 248879)
[PASS] test_DepositFor() (gas: 80656)
[PASS] test_RevertIf_DepositIsZero() (gas: 14766)
[PASS] test_RevertIf_SetUnaccountedGasTooHigh() (gas: 13666)
[PASS] test_RevertIf_TriesWithdrawToWithoutRequest() (gas: 13027)
[PASS] test_SetUnaccountedGas() (gas: 21528)
[PASS] test_addNewSigner() (gas: 42647)
[PASS] test_executeWithdrawalRequest_Fails_with_NoRequestSubmitted() (gas: 19151)
[PASS] test_executeWithdrawalRequest_Happy_Scenario() (gas: 163408)
[PASS] test_executeWithdrawalRequest_Reverts_If_Withdraws_TooSoon() (gas: 169019)
[PASS] test_removeSigner() (gas: 33860)
[PASS] test_submitWithdrawalRequest_Fails_with_ZeroAddress() (gas: 14230)
[PASS] test_submitWithdrawalRequest_Fails_with_ZeroAmount() (gas: 16349)
Suite result: ok. 13 passed; 0 failed; 0 skipped; finished in 12.89ms (16.79ms CPU time)

Ran 21 tests for
test/foundry/unit/concrete/TestStartaleManagedPaymaster.t.sol:TestStartaleManagedPaymaster
[PASS] test_AddVerifyingSigner() (gas: 56721)
[PASS] test_CheckInitialPaymasterState() (gas: 28415)
[PASS] test_Deploy() (gas: 1126645)
[PASS] test_Failure_TwoStepOwnershipTransferExpired() (gas: 48109)
[PASS] test_Failure_TwoStepOwnershipTransferWithdrawn() (gas: 35453)
[PASS] test_ParsePaymasterAndData() (gas: 59660)
[PASS] test_Receive() (gas: 18407)
[PASS] test_RemoveVerifyingSigner() (gas: 28574)
[PASS] test_RevertIf_AddVerifyingSignerToZeroAddress() (gas: 28953)
[PASS] test_RevertIf_DeployWithSignerAsContract() (gas: 276390)
```

```
[PASS] test_RevertIf_DeployWithSignerSetToZero() (gas: 99295)
[PASS] test_RevertIf_UnauthorizedOwnershipTransfer() (gas: 13989)
[PASS] test_RevertIf_ValidatePaymasterUserOpWithIncorrectSignatureLength() (gas: 167691)
[PASS] test_RevertIf_ValidatePaymasterUserOpWithInsufficientDeposit() (gas: 91611)
[PASS] test_RevertIf_WithdrawErc20ToZeroAddress() (gas: 537901)
[PASS] test_RevertIf_WithdrawEthExceedsBalance() (gas: 26275)
[PASS] test_Revert_DirectOwnershipTransfer() (gas: 16621)
[PASS] test_Success_TwoStepOwnershipTransfer() (gas: 38901)
[PASS] test_ValidatePaymasterEntireOp() (gas: 179980)
[PASS] test_WithdrawErc20() (gas: 554967)
[PASS] test_WithdrawEth() (gas: 27404)
Suite result: ok. 21 passed; 0 failed; 0 skipped; finished in 17.73ms (6.69ms CPU time)

Ran 43 tests for test/foundry/unit/concrete/TestSponsorshipPaymaster.t.sol:TestSponsorshipPaymaster
[PASS] test_AddVerifyingSigner() (gas: 57571)
[PASS] test_CheckInitialPaymasterState() (gas: 38904)
[PASS] test_Deploy() (gas: 1960032)
[PASS] test_DepositFor() (gas: 84576)
[PASS] test_Failure_TwoStepOwnershipTransferExpired() (gas: 48811)
[PASS] test_Failure_TwoStepOwnershipTransferWithdrawn() (gas: 36045)
[PASS] test_ParsePaymasterAndData() (gas: 66312)
[PASS] test_Receive() (gas: 18553)
[PASS] test_RemoveVerifyingSigner() (gas: 29160)
[PASS] test_RevertIf_AddVerifyingSignerToZeroAddress() (gas: 29935)
[PASS] test_RevertIf_DeployWithFeeCollectorAsContract() (gas: 305822)
[PASS] test_RevertIf_DeployWithFeeCollectorSetToZero() (gas: 129524)
[PASS] test_RevertIf_DeployWithSignerAsContract() (gas: 280166)
[PASS] test_RevertIf_DeployWithSignerSetToZero() (gas: 102873)
[PASS] test_RevertIf_DeployWithUnaccountedGasCostTooHigh() (gas: 135125)
[PASS] test_RevertIf_DepositCalled() (gas: 9814)
[PASS] test_RevertIf_DepositForZeroAddress() (gas: 16826)
[PASS] test_RevertIf_DepositForZeroValue() (gas: 12777)
[PASS] test_RevertIf_SetFeeCollectorToZeroAddress() (gas: 14597)
[PASS] test_RevertIf_SetUnaccountedGasToHigh() (gas: 14823)
[PASS] test_RevertIf_TriesWithdrawToWithoutRequest() (gas: 14115)
[PASS] test_RevertIf_UnauthorizedOwnershipTransfer() (gas: 14669)
[PASS] test_RevertIf_UnauthorizedSetFeeCollector() (gas: 12894)
[PASS] test_RevertIf_ValidatePaymasterUserOpWithIncorrectSignatureLength() (gas: 205099)
[PASS] test_RevertIf_ValidatePaymasterUserOpWithInsufficientDeposit() (gas: 95210)
[PASS] test_RevertIf_ValidatePaymasterUserOpWithInvalidPriceMarkUp() (gas: 172984)
[PASS] test_RevertIf_WithdrawErc20ToZeroAddress() (gas: 538535)
[PASS] test_RevertIf_WithdrawEthExceedsBalance() (gas: 26597)
[PASS] test_Revert_DirectOwnershipTransfer() (gas: 17053)
[PASS] test_SetFeeCollector() (gas: 27105)
[PASS] test_SetUnaccountedGas() (gas: 25882)
[PASS] test_Success_TwoStepOwnershipTransfer() (gas: 39564)
[PASS] test_ValidatePaymasterAndPostOpWithPriceMarkup() (gas: 248978)
[PASS] test_ValidatePaymasterAndPostOpWithoutPriceMarkup() (gas: 250510)
[PASS] test_WithdrawErc20() (gas: 555199)
[PASS] test_WithdrawEth() (gas: 27770)
[PASS] test_depositFor_RevertsIf_DepositIsLessThanMinDeposit() (gas: 23152)
[PASS] test_executeWithdrawalRequest_Fails_with_NoRequestSubmitted() (gas: 20151)
[PASS] test_executeWithdrawalRequest_Withdraws_WhateverIsLeft() (gas: 249786)
[PASS] test_submitWithdrawalRequest_Fails_If_not_enough_balance() (gas: 86315)
[PASS] test_submitWithdrawalRequest_Fails_with_ZeroAddress() (gas: 14794)
[PASS] test_submitWithdrawalRequest_Fails_with_ZeroAmount() (gas: 17965)
[PASS] test_submitWithdrawalRequest_Happy_Scenario() (gas: 139786)
Suite result: ok. 43 passed; 0 failed; 0 skipped; finished in 18.17ms (9.34ms CPU time)

Ran 24 tests for test/foundry/unit/concrete/TestTokenPaymaster.t.sol:TestTokenPaymaster
[PASS] testCounter() (gas: 3129)
[PASS] testToken() (gas: 2799)
[PASS] test_AddVerifyingSigner() (gas: 58297)
[PASS] test_Allow_Treasury_ToBeSelf() (gas: 2698202)
[PASS] test_Deploy_STPM() (gas: 2703001)
[PASS] test_Deposit() (gas: 58518)
[PASS] test_DoesNotRevertIf_Mismatching_Oracle_Decimals_Independent_Mode() (gas: 313020)
[PASS] test_ParsePaymasterAndDataForExternalMode() (gas: 62680)
[PASS] test_RemoveVerifyingSigner() (gas: 29607)
[PASS] test_RevertIf_DeployWithSignerSetToZero() (gas: 113068)
[PASS] test_RevertIf_FeeTreasuryIsZero() (gas: 236054)
[PASS] test_RevertIf_InvalidSignature_ExternalMode() (gas: 233935)
```

```
[PASS] test_RevertIf_InvalidTokenAddress_Independent_Mode() (gas: 671663)
[PASS] test_RevertIf_UnaccountedGasTooHigh() (gas: 14719)
[PASS] test_RevertIf_UserDoesNotHaveEnoughBalance_Any_Mode() (gas: 216944)
[PASS] test_Revert_PostOp_If_PriceExpired() (gas: 237257)
[PASS] test_SetPriceMarkupTooHigh() (gas: 19205)
[PASS] test_Success_TokenPaymaster_ExternalMode_WithPremium() (gas: 299128)
[PASS] test_Success_TokenPaymaster_ExternalMode_WithoutPremium() (gas: 302345)
[PASS] test_Success_TokenPaymaster_IndependentMode_WithPremium() (gas: 324559)
[PASS] test_Success_TokenPaymaster_IndependentMode_WithoutPremium() (gas: 317759)
[PASS] test_Success_test_UpdateNativeOracleConfig() (gas: 20397)
[PASS] test_WithdrawERC20() (gas: 68530)
[PASS] test_WithdrawTo() (gas: 55138)
Suite result: ok. 24 passed; 0 failed; 0 skipped; finished in 45.17ms (18.99ms CPU time)

Ran 4 tests for test/foundry/unit/fuzz/TestFuzz_SponsorshipPaymaster.t.sol:TestFuzz_SponsorshipPaymaster
[PASS] testFuzz_DepositFor(uint256) (runs: 1000, μ: 85528, ~: 85528)
[PASS] testFuzz_Receive(uint256) (runs: 1000, μ: 19146, ~: 19146)
[PASS] testFuzz_WithdrawErc20(address,uint256) (runs: 1000, μ: 554580, ~: 555038)
[PASS] testFuzz_WithdrawEth(uint256) (runs: 1000, μ: 27275, ~: 27275)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 202.38ms (305.03ms CPU time)

Ran 5 tests for test/foundry/unit/concrete/TestTokenPaymaster.Soneium.t.sol:TestTokenPaymasterSoneium
[PASS] testCounter() (gas: 2777)
[PASS] test_Deploy_STPM_Soneium() (gas: 2708647)
[PASS] test_Deposit_Soneium() (gas: 58051)
[PASS] test_Success_TokenPaymaster_IndependentMode_WithoutPremium_Soneium() (gas: 559169)
[PASS] test_WithdrawTo_Soneium() (gas: 54997)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 11.84s (4.85s CPU time)

Ran 6 test suites in 11.85s (12.13s CPU time): 110 tests passed, 0 failed, 0 skipped (110 total tests)
```

Code Coverage

Test coverage data was obtained with `forge coverage`. It is recommended to increase the code coverage to 100%.

Fix-Review Update: There have been no notable changes to the test coverage.

File	% Lines	% Statements	% Branches	% Funcs
src/base/BasePaymaster.sol	60.00% (12/20)	65.22% (15/23)	25.00% (1/4)	66.67% (10/15)
src/lib/MultiSigners.sol	76.92% (20/26)	82.14% (23/28)	42.86% (3/7)	100.00% (5/5)
src/lib/TokenPaymasterParserLib.sol	80.00% (12/15)	81.25% (13/16)	0.00% (0/3)	100.00% (3/3)
src/sponsorship/SponsorshipPaymaster.sol	87.22% (116/133)	88.02% (147/167)	67.65% (23/34)	91.30% (21/23)
src/sponsorship/StartaleManagedPaymaster.sol	92.86% (26/28)	93.94% (31/33)	66.67% (4/6)	100.00% (9/9)
src/token/startale/PriceOracleHelper.sol	61.22% (30/49)	66.13% (41/62)	15.38% (2/13)	75.00% (6/8)
src/token/startale/StartaleTokenPaymaster.sol	68.71% (101/147)	73.77% (135/183)	24.32% (9/37)	78.26% (18/23)
src/utils/SoladyOwnable.sol	25.00% (1/4)	25.00% (1/4)	0.00% (0/1)	100.00% (1/1)
Total	75.36% (318/422)	78.68% (406/516)	40.00% (42/105)	83.91% (73/87)

Changelog

- 2025-04-15 - Initial report
- 2025-04-30 - Final report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT